

UNIVERSIDAD DEL BIOBIO

MICROPROCESADORES 8031-8032-8051-8052

La familia MCS-51 a la cual pertenece el procesador 8031 es una de las familias de procesadores que han tenido bastante aceptación en el mundo industrial. Los procesadores de esta línea se encuentran en muchos de los PLC de marca y su uso como microcontrolador aún está vigente, a pesar que INTEL ha dejado de fabricarlo.

Entre las características de este procesador se pueden mencionar:

- Bus de 8 bits
- Acceso a 64 K ROM
- Acceso a 64K de RAM
- RAM interna
- 2 (3) Timers/Counters internos
- UART incorporada

Proposición de Circuitos básicos

Circuito 1/4:

La figura 1/3 muestra una conexión típica que incluye un cristal de 12 MHz, botón RESET y un 74LS573 (idéntico al 74LS373 sólo con una diferente asignación de pines). El 573 (373) permite separar el bus de datos BD del bus de direcciones bajo, BAL. Nótese que este bus multiplexado corresponde a la puerta 0, la que se pierde como tal y trabaja como bus. Nótese la habilitación del 573(373) con la señal ALE.

Circuito 2/4:

Es necesario mencionar que por diseño posee 4 puertas de 8 bits, pero sólo una queda disponible cuando se conecta ROM o RAM externa. Esto lleva a que deba utilizarse un esquema de MAPEO de MEMORIA para incorporar más puertas al sistema. Por comodidad, este mapeo se realiza en los sectores altos de la memoria, como se indica en la figura 2/3 del presente manual. Alli se muestra la decodificación propuesta para 8 puertas de entrada mapeadas desde FFF8h a FFFFh, al igual que para 8 puertas de salida. Esta es una proposición, existiendo un sinnúmero de circuitos que pueden realizar esta tarea, incluyendo el uso de PAL. Debe notarse la generación de la señal RAMDIS, activa baja que será utilizada en el circuito 3/3 para deshabilitar la RAM cuando exista acceso al sector FFF8h-FFFFh, evitando la colisión que producirían las puertas y la RAM si ambas estuvieren simultáneamente activadas.

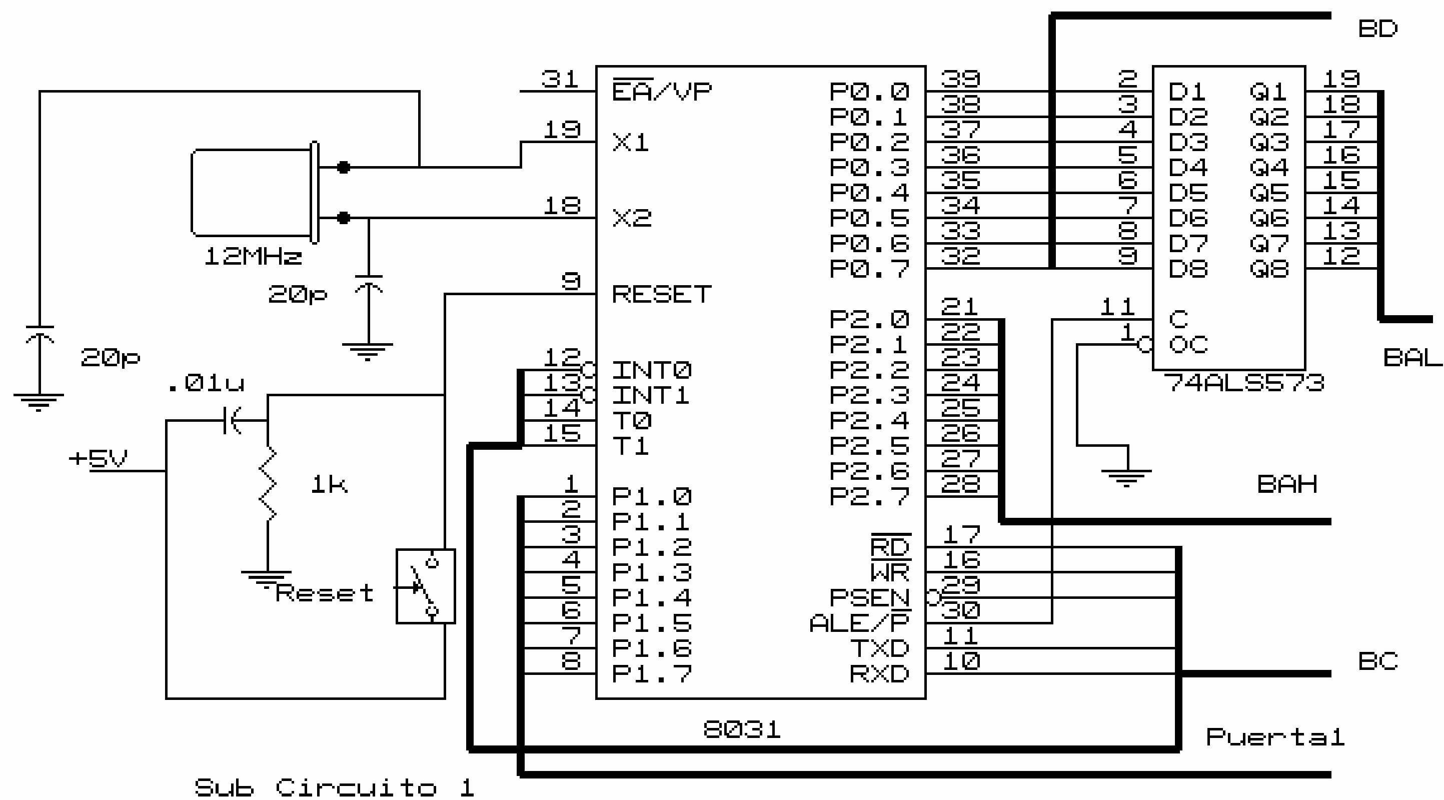
Circuito 3/4:

En este circuito se propone un esquema para conectar una EPROM 2764 (8Kx8) y una RAM 6264 (8K8). La dirección de la EPROM es la 0000, en tanto que la dirección de inicio de la RAM dependerá de la decodificación que se realice para generar la señal RAMSEL, activa baja.

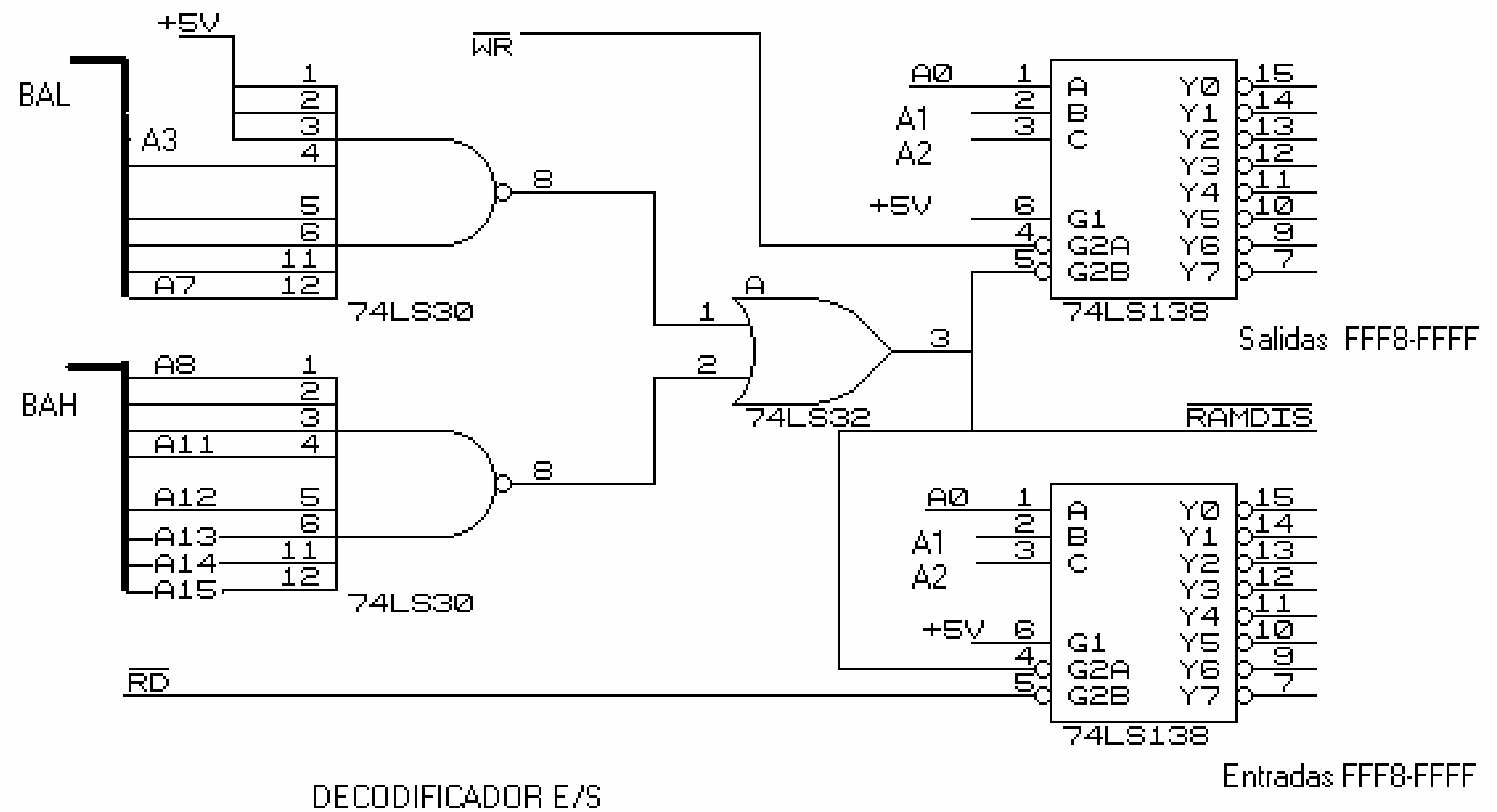
Circuito 4/4

Este circuito está propuesto para ser habilitado con alguna de las salidas Y0 a Y7 del 74ls138. Es un circuito que requiere de un software que transforme el número o carácter a imprimir en los segmentos que deben iluminarse. Se sugiere ver la rutina **Display** que está más adelante.

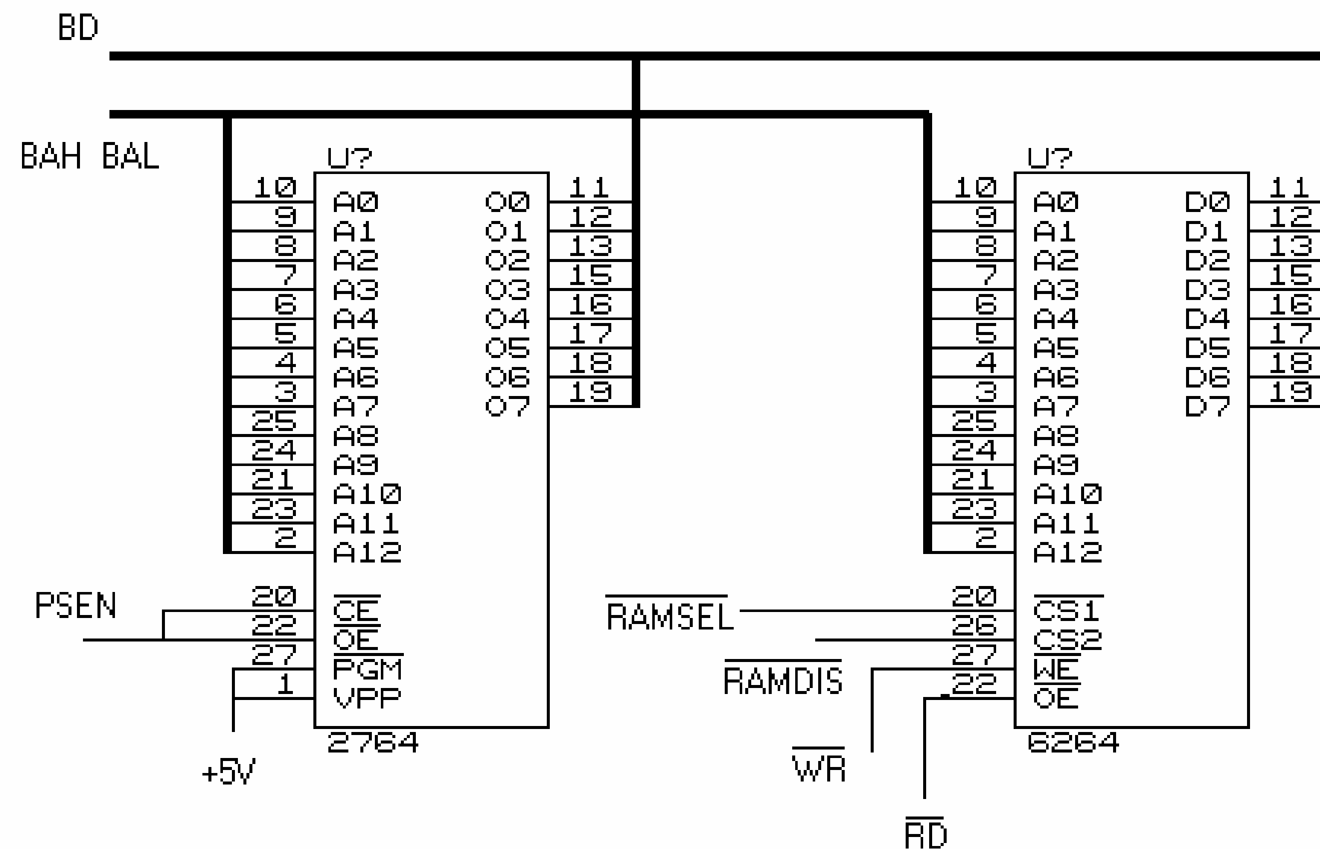




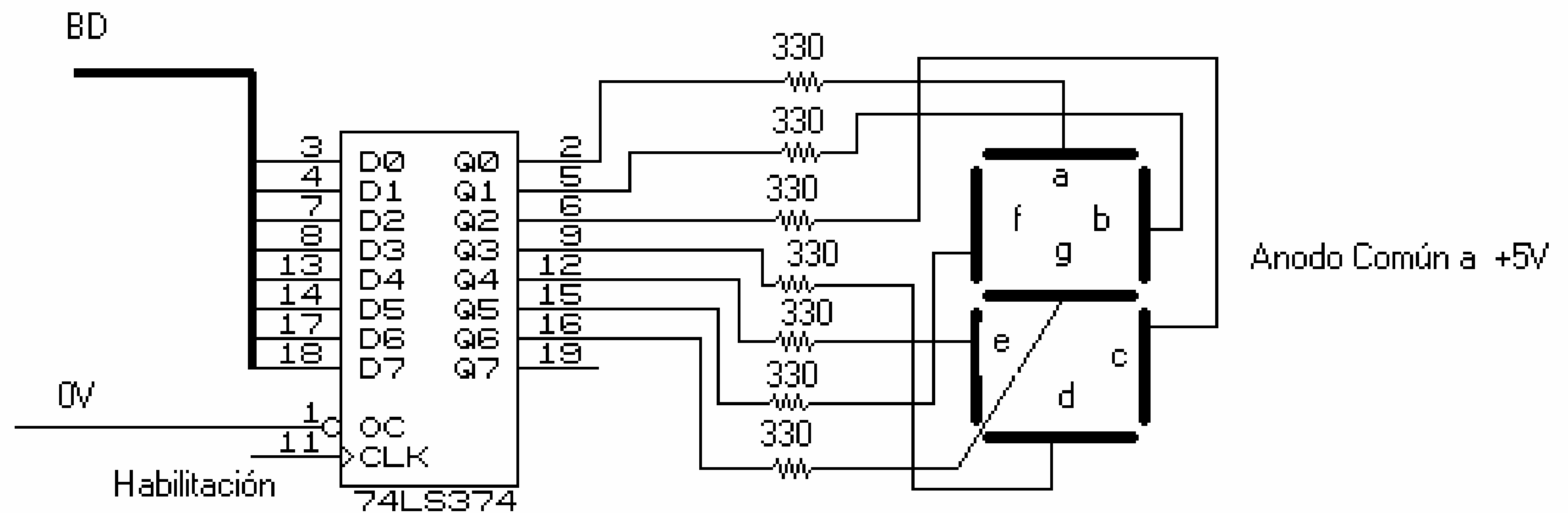
CIRCUITO PARA SEPARAR BUS DE DIRECCIONES (BAH-BAL) DEL BUS DE DATOS (BD) 1/4



Circuito decodificador E/S
2/4



Circuito para conexión de ROM y RAM. Señal RAMDIS proviene de circuito anterior
3/4



Puerta de Salida con Display 7 segmentos Anodo común
4/4

Acerca del Software
Programación:

La programación se hace escribiendo el programa en un procesador de texto simple, por ejemplo EDIT, NOTEPAD. El archivo se debe nombrar con la extensión ASM. Luego se debe ENSAMBLAR con el programa ASM8031 utilizando el switch "o" Las instrucciones son las que aparecen en la sección 6 "Programmers guide and instruction set".

Luego de ensamblado con la opción "o", la salida es un archivo .obj, en formato HEX INTEL. Este formato es reconocido por los programadores de EPROM, por lo que no hay más trámites que hacer que grabar la EPROM.

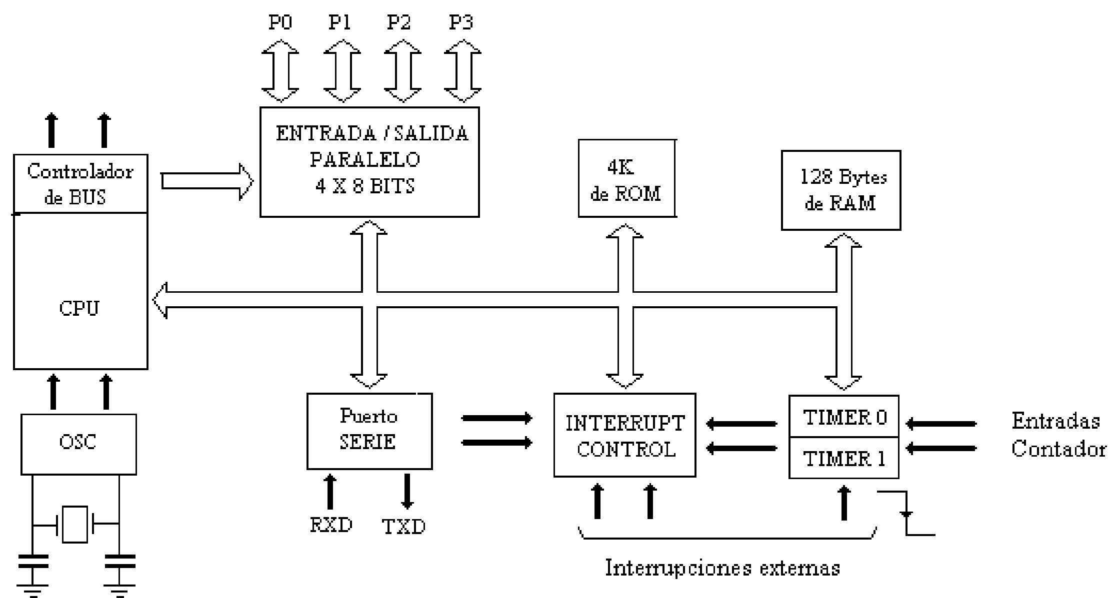
FAMILIA DE MICROCONTROLADORES INTEL 8X52 y 8X51

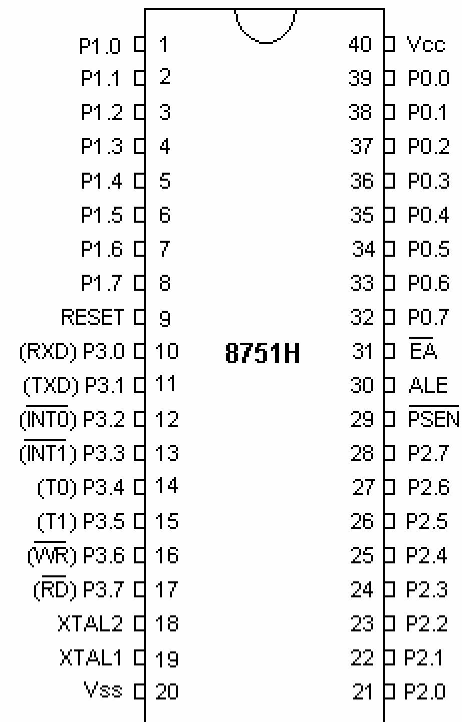
Versión con ROM	Versión sin ROM	Versión con EPROM	ROM Bytes	RAM Bytes	16-bit Timers	Tecnología
8051	8031	8751	4K	128	2	HMOS
8051AH	8031AH	8751H	4K	128	2	HMOS
8052AH	8032AH	8752BH	8K	256	3	HMOS
80C51BH	80C31BH	87C51	4K	128	2	CHMOS

Características:

- CPU de 8 bits.
- Procesador booleano (operación sobre bits).
- 4 puertos de 8 bits.

- Para el 8051, 128 bytes de memoria interna RAM útil para el usuario y 256 bytes en total considerando el área de los registros especiales (SFR).
- Espacio de memoria de 64K para programa externo.
- Espacio de memoria de 64K para datos externos.
- El 8051 posee 2 contadores-temporizadores (*timers*).
- Comunicación asincrónica *full-duplex*.
- 5 fuentes de interrupciones con niveles de prioridad (para el 8051):
  - 2 interrupciones externas.
  - 2 interrupciones de los *timers*.
  - 1 interrupción de la comunicación serial.
- Oscilador interno.

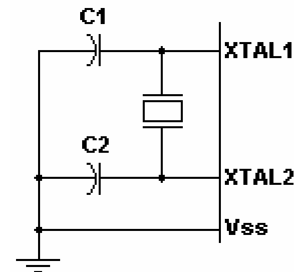

P0
P1
P2
P3
ENTRADA / SALIDA
PARALELO
4 X 8 BITS
Controlador
de BUS
4K
de ROM
128 Bytes
de RAM
CPU
OSC
Puerto
SERIE
RXD
TXD
INTERRUPT
CONTROL
TIMER 0
TIMER 1
Entradas
Contador
Interrupciones externas



- **Puertos:** Esta familia tiene 4 puertos bidireccionales (P0, P1, P2 y P3), que se pueden programar tanto como entrada o como salida. P0 tiene un *fanout* de 8 (TTL), mientras que el de los otros tres puertos es solamente 4.
- **P0:** Multiplexa en el tiempo, la parte baja del bus de direcciones durante el acceso a la memoria externa de programas y datos, y el bus de datos. También este puerto recibe los datos almacenados en la memoria externa.
- **P2:** Emite la parte alta del bus de direcciones, pero también puede ser utilizado como un puerto de propósito general.
- **P3:** Además de servir como puerto de propósito general, sus pines cumplen algunas funciones especiales como:

Pines	Función Alternativa
P3.0	RXD (entrada puerto serie)
P3.1	TXD (salida puerto serie)
P3.2	$\overline{INT0}$ (Interrupción 0 externa)
P3.3	$\overline{INT1}$ (Interrupción 1 externa)
P3.4	T0 (Entrada externa Timer 0)
P3.5	T1 (entrada externa Timer 1)
P3.6	$\overline{WR}$ (Autorización escritura en memoria de datos externa)
P3.7	$\overline{RD}$ (Autorización lectura en memoria de datos externa)

- **ALE:** (Address Latch Enable), es un pulso emitido por el microntrolador para enclavar el "Byte bajo" del bus de direcciones en el acceso a la memoria externa. ALE se emite con una frecuencia de 1/6 de la frecuencia de emisión del reloj.
- **PSEN($\overline{PSEN}$):** (Program Store Enable) es la señal de strobe para leer en la memoria de programa externo. La memoria externa, tiene dos modalidades, de programa y de datos. Para diferenciarlas, utiliza la señal PSEN. PSEN no se activa cuando se está ejecutando el programa de la ROM y EPROM interna.
- **EA($\overline{EA}$):** (External Access), cuando se mantiene a nivel alto, se ejecuta solo el programa de la ROM interna, salvo que el contador se exceda de la capacidad de ésta. Si se mantiene a nivel bajo, se ejecuta el programa de la memoria externa, independientemente de la dirección del programa.
- **XTAL1 y XTAL2:** Estos pines son la entrada y salida, respectivamente de un amplificador inversor que puede ser configurado para su uso como un *chip* oscilador.



- **RESET:** Señal de inicialización del sistema. Un *reset* interno al sistema se produce cuando se pone el pin RESET a nivel alto durante un cierto tiempo.

MEMORIA

- **Programas:**
  - Sólo puede ser leída y tiene un máximo de 64K.
  - La señal que autoriza la lectura de la memoria del programa externa es $\overline{PSEN}$.
  - $\overline{EA} = Vcc$ en el caso del 8052, PC recorre las posiciones de memoria del 0000H al 1FFFH dentro del microprocesador y desde la 2000H a la FFFFH, en la memoria externa.
  - $\overline{EA} = GND$ PC siempre busca en la memoria externa.

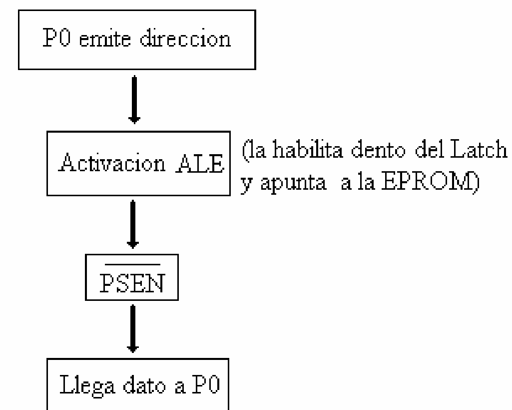
- **Datos:**
  - Admite operaciones de lectura y de escritura.
  - Puede ser interna o externa, y su capacidad de direccionamiento es de 64K.

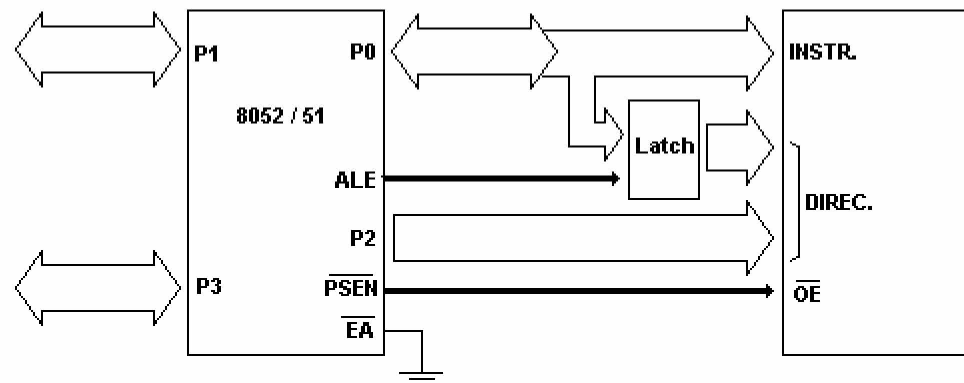
- Las señales $\overline{RD}$ y $\overline{WD}$ son generadas por la CPU para manejar operaciones de lectura y escritura de la memoria externa.

Memoria de programas

0000H	Reset
0003H	Ext 0 (IE0)
000BH	Timer 0 (TF0)
0013H	Ext1 (IE1)
001BH	Timer 1 (TF1)
0023H	Puerto Serie (RI y TI)

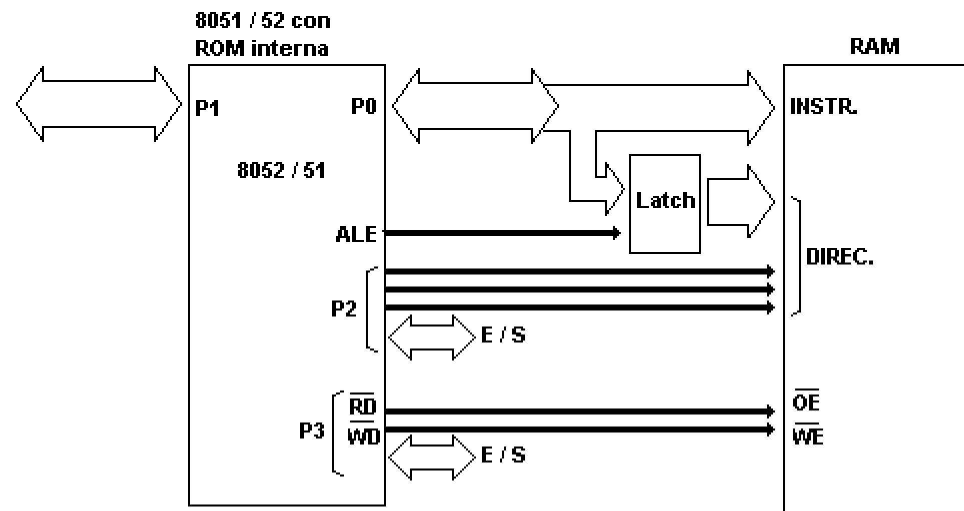
Ciclo de lectura del programa externo





Memoria de Datos

El ciclo de lectura y escritura de esta memoria, no utiliza $\overline{PSEN}$, sino que $\overline{RD}$ y $\overline{WD}$.



Mapa de memoria de datos

La memoria interna de datos, esta dividida en 2 bloques:

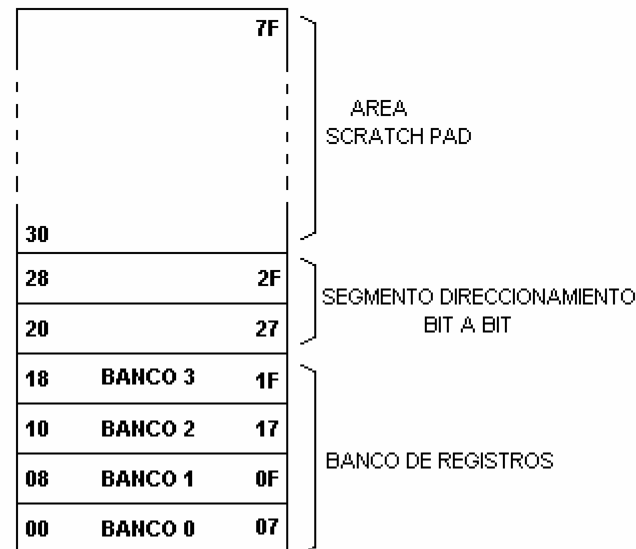
- 128 bytes bajos con direccionamiento directo e indirecto.
- Espacio ocupado por los Registros de funciones especiales (SFR).

Área de direccionamiento directo e indirecto (00H al 7FH)

- Banco de Registros (0, 1, 2 y 3), con R0 a R7 por cada banco.
  - Son 32 bytes ubicados desde 00H a 1FH.
  - Cada vez que se presiona RESET se vuelve al Banco 0 por defecto.
  - La selección de que Banco se desea utilizar, se hace en el registro PSW.

- Subárea de direccionamiento bit a bit
  - Longitud de 16 bytes (20H a 2FH).
  - Cada uno de los 128 bits se puede direccionar directamente por su dirección (00H – 7FH), o por el Byte que lo contiene (20H a 2FH), ej: 20.0, 20.7, 21.5, etc.
  - También pueden ser direccionados como Bytes.

- Subárea Scratch Pad.
  - Ubicada entre las direcciones 30H a la 7FH.
  - Es una memoria de trabajo del usuario, de poca capacidad, donde se almacenan por ejemplo las variables creadas en el programa.



Área de Registros y Funciones especiales

Símbolo	Nombre	Dirección
*ACC	Accumulator	**0E0H**
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	--
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer / Counter Mode Control	89H
*TCON	Timer / Counter Control	88H
*+T2CON	Timer / Counter 2 Control	0C8H
TH0	Timer / Counter 0 High Byte	8CH
TL0	Timer / Counter 0 Low Byte	8AH
TH1	Timer / Counter 1 High Byte	8DH
TL1	Timer / Counter 1 Low Byte	8BH
+TH2	Timer / Counter 2 High Byte	0CDH
+TL2	Timer / Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H

SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

* : Direccionamiento bit a bit.
+: Sólo para el 8052.

Descripciones de algunos de los registros:

Acc: Es un registro de propósito general y por su frecuencia de uso, el más importante.

B: Usado en multiplicaciones y divisiones, pero también puede ser usado como propósito general.

PSW: Contiene información del estado de la CPU en cada ciclo de instrucción.

DPL y DPH: Su propósito principal es contener la dirección del puntero de datos. Puede ser manipulado como un registro de 16 bits (**DPTR**) o como dos registros independientes de 8 bits.

IE: En este registro se habilitan o deshabilitan interrupciones según lo desee el usuario.

IP: Establece las prioridades entre las interrupciones habilitadas en IE.

Programación de la Familia 51

Modos de direccionamiento

- Direccionamiento Directo:

- Solo la RAM interna y la zona de SFR se puede direccionar de esta manera.
- El operando que especifica la instrucción es una dirección de 8 bits.

Ej:

ADD A, 3BH

- Direccionamiento Indirecto:
  - Es un registro que contiene la dirección del operando.
  - Tanto las memorias RAM interna como externa, se pueden direccionar de esta manera.
  - Los registros para direccionar sobre el mapa de 8 bits pueden ser R0 y R1 del banco seleccionado, o el Stack Pointer.

  Ej:

  ADD A, @R0

- Direccionamiento por Registro:
  - Dependiendo de cual banco de registros esté seleccionado, habrán de R0 a R7 registros disponibles a los cuales se les puede asignar el valor contenido en un registro, como lo es el Acumulador.

  Ej:

  MOV R0, A

- Direccionamiento Implícito:
  - Algunas instrucciones indican explícitamente sobre cual registro se va a operar, sin que sea necesario especificar el operando.

  Ej:

  INC A
  INC DPTR

- Direccionamiento Inmediato:
  - Al código de operación, le sigue una constante en la memoria de programas.

  Ej:

  MOV A, #255
  MOV A, #FFH
  MOV A, #11111111B

- Direccionamiento Indexado:
  - Sólo es posible en la memoria de programas y solo permite la lectura.
  - Es utilizado para lecturas de tablas.
  - Un registro base de 16 bits (DPTR o PC) apunta a la base de la tabla y el contenido del acumulador es el *offset* que permite acceder a la lectura de esa posición de la tabla.

  Ej:

  MOVC A, @A+DPTR

Tipos de instrucciones

- Instrucciones Aritméticas.

  ADD A, <byte>
  ADDC A, <byte>
  SUBB A, <byte>
  INC A
  INC <byte>
  INC DPTR

DEC	A
DEC	<byte>
MUL	AB
DIV	AB
DA	A

- Instrucciones lógicas.

ANL	A, <byte>
ANL	<byte>, A
ANL	<byte>, #data
ORL	A, <byte>
ORL	<byte>, A
ORL	<byte>, #data
XRL	A, <byte>
XRL	<byte>, A
XRL	<byte>, #data
CLR	A
CPL	A
RL	A
RLC	A
RR	A
RRC	A
SWAP	A

- Instrucciones de transferencia de datos.

Sobre RAM interna:

MOV	A, <src>
MOV	<dest>, A
MOV	<dest>, <src>
MOV	DPTR, #data 16
PUSH	<src>
POP	<dest>
XCH	A, <byte>
XCHD	A, @Ri

Sobre RAM externa (8 bits):

MOVX	A, @Ri
MOVX	@Ri, A

- Instrucciones booleanas.

ANL	C, bit
ANL	C, /bit
ORL	C, bit
ORL	C, /bit
MOV	C, bit
MOV	bit, C
CLR	C
CLR	bit
SETB	C
SETB	bit
CPL	C

CPL bit
JC rel
JNC rel
JB bit, rel
JNB bit, rel
JBC bit, rel

- Instrucciones de salto.

Instrucciones de salto incondicional:

JMP addr
JMP @A + DPTR
CALL addr
RET
RETI
NOP

Instrucciones de salto condicional:

JZ rel
JNZ rel
DJNZ <byte>, rel
CJNE A, <byte>, rel
CJNE <byte>, #data, rel

Comenzando la Programación

- Antes de comenzar a pensar en la lógica del programa, hay ciertos pasos que se deben cumplir, como por ejemplo, especificar la dirección de memoria a partir de la cual se comenzará a escribir el programa, la cual tiene que encontrarse después de las zonas predefinidas como lo son las asignadas para interrupciones.

- Si se desean utilizar valores diferentes a los que vienen por defecto de los registros IE, IP, SMOD, etc. se debe tener en consideración, a la hora de la elección, los seteos de cada bit (esto se hace al principio de cada programa).

SUBRUTINAS DE UTILIDAD PARA LA FAMILIA DE PROCESADORES 8031-51

A continuación se muestran algunos programas útiles que pueden ser adaptados para incluirlos según sea necesario.

Preparación del TIMER para trabajo con la UART interna:

```
; En este ejemplo de usa un cristal de 11.059 MHz y el TIMER 1.
; El valor 0FDh cargado en th1 provee 9600 Baudios.
; Para detalles o cálculo de otra velocidad, con otros crsitales se debe consultar página 7-17

        mov   tmod, #20h        ; timer 1 para autorecarga  - modo 2
        mov   tcon, #41h        ; run timer 1 and set edge trig ints (parte el timer 1)
        mov   th1,  #0fdh       ; set timer 1 for 9600 baud with xtal=11.059MHz
        mov   scon, #50h        ;  serial control reg para 8 bit data y modo 1 (ver página 7-16)
```

Subrutina mdelay – retardo de 1 milisegundo, basada en un cristal de 12 MHz

```
;   Retarda  998 microsegundos – y agrega 2 microsegundos por la llamada
;  input    : none  output   : none
;  destroys : nothing - uses a
; 100h-a6h=5ah=(90)decimal  90 * 11 = 990  + 8 da 998 microsegundos (ciclos)

mdelay: push acc        ;    2
        mov   a, #0a6h  ;    1
md_olp: inc   a         ;    1
        nop             ;    1  |
        nop             ;    1  |
        nop             ;    1  |
        nop             ;    1  |
        nop             ;    1  |- 11  ciclos
        nop             ;    1  |
        nop             ;    1  |
        nop             ;    1  |
        jnz   md_olp    ;    2  / 1 ciclo si no hay jump

        nop             ;    1
        pop   acc       ;    2
```

```
    ret             ;    2
```

Rutina que envía caracter que está en el acc por la puerta serial : sndchr

```
sndchr:
         clr  scon.1            ; clear  tx  buffer full flag.
         mov  sbuf,a            ; pone  chr  en sbuf, esta es la transmisión real del caracter
txloop:  jnb  scon.1, txloop    ; espera hasta que el chr esté enviado (conversión paralelo serie a xxx baudios)
         ret                    ;retorna o sigue para otro lado pues ya salió el chr
```

Rutina que recibe un carácter que llega a la puerta serial y lo deja en el acc

```
getchrx:
         jnb  ri, getchrx  ; espera hasta recibir un caracter
         mov  a,  sbuf     ; pasa el carácter al acumulador
         clr  ri           ; clear serial status bit y deja listo a recibir otro
         ret
```

Rutina display – muestra un dígito de 0 a 9 y de A hasta F para display de 7 segmentos

```
;  input    : nibble en accumulator
;  output   : 7-segment pattern in accumulator
;           (acc.0 es el segmento a, acc.6 es el segmento g y el hardware debe estar b0= a  ....b6=g
;  destuye: a
; ---------------------------------------------------
display:
    inc  a
    movc a, @a+pc
    ret
    db  0c0h  ; 0
    db  0f9h  ; 1
    db  0a4h  ; 2
    db  0b0h  ; 3
    db  99h  ; 4
```

```
db  92h  ; 5
db  82h  ; 6
db  0f8h  ; 7
db  80h  ; 8
db  90h  ; 9
db  88h  ; a
db  83h  ; b
db  0c6h  ; c
db  0a1h  ; d
db  86h  ; e
db  8eh  ; f
```

Se sugiere estudiar el fucionamiento de la instrucción **movc a, @a+pc.** Nótese que el inicio de la rutina es con la instrucción **inc a**, esto debido a que la tabla comienza después de la instrucción **ret.**

Analice si este programa trabaja con display de ánodo o cátodo común. ¿Podría ampliarlo para mostrar otros caracteres, por ejemplo H, L, n, u, i, o, p, r, ?

Subrutinas matemáticas:

A continuación se muestra una serie de subrutinas para resolver algunas operaciones matemáticas tales como sumas, restas, multiplicaciones y divisiones en 8, 16 y 32 bit.

```
; subroutine TC2AV_SB8
; 8-Bit Two's Complement to Absolute Value / Sign Bit Conversion
;
; input:     internal register XT holds a signed byte in two's
;            complement convension
; output:    internal register XA holds the absolute value of the
;            signed byte, addressable bit XS holds the sign (XS is
;            set if XT is a negative number)
; destroys:  a
;===============================================================
TC2AV_SB8:
```

```
    mov     a, XT          ; read X into accumulator
    jb     acc.7, Xneg     ; X is negative if bit 7 is 1

    mov     XA, XT         ; else, XT is the absolute value
    clr     XS             ; clear sign bit for 'positive'
    ret                    ; done
Xneg:
    cpl     a              ; X is negative so find absolute value
    inc     a              ; XA = copmlement(XT)+1
    mov     XA, a          ; save absolute value
    setb    XS             ; set sign bit for 'negative'
    ret


;====================================================================
; subroutine AV_SB2TC8
; 8-Bit Absolute Value / Sign Bitto Two's Complement Conversion
;
; input:     internal register XA holds the absolute value of the
;          signed byte, addressable bit XS holds the sign (XS is
;          set if XT is a negative number)
; output:     internal register XT holds a signed byte in two's
;          complement convension.  the addressable bit XOV is
;          set if the byte is out of range (-128 to 127).
; destroys:     a
;====================================================================
AV_SB2TC8:
    mov     a, XA              ; get absolute value
    jnb     acc.7, range_OK     ; see if MSB is 0
    cjne     a, #80h, range_error
    jb     XS, range_OK     ; also error if X is positive

range_error:
    setb     XOV          ; set overflow bit
    ret
```

```
range_OK:
    clr     XOV          ; no overflow error
    jb     XS, Xneg1     ; check sign
    mov     XT, XA     ; if positive, XT is the same as XA
    ret               ; done
Xneg1:
    mov     a, XA          ; if X is negative, get its absolute value
    cpl     a          ; complement absolute value of X
    inc     a          ; XT = copmlement(XA)+1
    mov     XT, a          ; save two's complement representation
    ret


;=================================================================
; subroutine TC2AV_SB16
; 16-Bit Two's Complement to Absolute Value / Sign Bit Conversion
;
; input:     internal registers XTH and XTL hold a signed word in
;         two's complement convension
; output:     internal registers XAH and XAL hold the absolute
;         value of the signed word, addressable bit XS holds
;         the sign (XS is set if XT is a negative number)
; destroys:     a
;=================================================================
TC2AV_SB16:
    mov     a, XTH       ; read X high byte into accumulator
    jb      acc.7, Xneg2 ; X is negative if bit 7 is 1

    mov     XAH, XTH     ; else, XT is the absolute value
    mov     XAL, XTL
    clr     XS          ; clear sign bit for 'positive'
    ret                 ; done
Xneg2:
    setb    XS
    mov     a, XTL       ; X is negative
```

```
    cpl    a          ; find its absolute value
    inc    a          ; XA = copmlement(XT)+1
    mov    XAL,a      ; XAL is found
    jnz    XL_OK      ; if not 0, high byte is not incremented

    mov    a, XTH     ; else get high byte
    cpl    a          ; complement high byte...
    inc    a          ; ... and increment
    mov    XAH,a      ; store in XAH
    ret               ; done
XL_OK:
    mov    a, XTH     ; get high byte
    cpl    a          ; complement high byte - do not increment
    mov    XAH,a      ; store in XAH
    ret               ; done

;====================================================================
; subroutine AV_SB2TC16
; 16-Bit Absolute Value / Sign Bit to Two's Complement Conversion
;
; input:    internal registers XAH and XAL hold the absolute
;           value of the signed word, addressable bit XS holds
;           the sign (XS is set if XT is a negative number)
; output:   internal registers XTH and XTL hold a signed word in
;           two's complement convension.  the addressable bit XOV is
;           set if the byte is out of range (-32768 to 32767).
; destroys: a
;====================================================================
AV_SB2TC16:
    mov    a, XAH            ; get absolute value
    jnb    acc.7, range_OK1  ; see if MSB is 0
    clr    acc.7             ; see if X=-8000h
    orl    a, XAL            ; acc=0 only if X=8000
    jnz    range_error1      ; else error
    jb     XS, range_OK1     ; also error if X is positive
```

```
range_error1:
    setb    XOV         ; set overflow bit
    ret

range_OK1:
    clr    XOV         ; no overflow error
    jb    XS, Xneg2     ; check sign
    mov    XTH, XAH     ; if positive, XT is the same as XA
    mov    XTL, XAL
    ret             ; done
Xneg3:
    mov    a, XAL     ; if X is negative, get its absolute value
    cpl    a         ; complement
    inc    a         ; XT = copmlement(XA)+1
    mov    XTL, a
    jnz    XL_OK1     ; if not 0, high byte is not incremented

    mov    a, XAH     ; else get high byte
    cpl    a         ; complement high byte...
    inc    a         ; ... and increment
    mov    XTH,a      ; store in XTH
    ret             ; done
XL_OK1:
    mov    a, XAH     ; get high byte
    cpl    a         ; complement high byte - do not increment
    mov    XTH,a      ; store in XTH
    ret             ; done

;============================================================
; subroutine SMUL8
; 8-Bit Multiplication of Signed Integers in the Absolute Value
; / Sign Bit Format
;
; input:     internal registers XA and YA hold the absolute
;         values of the signed bytes X and Y.  addressable
;         bits XS and YS hold the sign bits, set if the
```

```
;          corresponding numbers are negative
; output:     internal registers ZAH and ZAL hold the absolute
;          value of the result Z=X´Y.  addressable bit ZS
;          holds the sign of the result, set if the result is
;          negative.
; destroys:     a
;=================================================================
SMUL8:
    mov     a, XA          ; read X and ...
    mov     b, YA          ; ... Y
    mul     ab             ; multiply X and Y
    mov     ZH, b          ; save result high ...
    mov     ZL, a          ; ... and low byte

    jb      XS, SM8_2      ; get sign of X
    jb      YS, SM8_1
    clr     ZS
    ret
SM8_1:
    setb    ZS
    ret
SM8_2:
    jb      YS, SM8_3
    setb    ZS
    ret
SM8_3:
    clr     ZS
    ret

;================================================================
```

; subroutine SDIV8
; 8-Bit Division of Signed Integers in the Absolute Value /
; Sign Bit Format

```
;
; input:     internal registers XA and YA hold the absolute
;          values of the signed bytes X and Y.  addressable
```

```
;         bits XS and YS hold the sign bits, set if the
;         corresponding numbers are negative
; output:    internal registers ZAQ and ZAR hold the quotient and
;         the remainder of the division Z=X¸Y.  addressable
;         bit ZS holds the sign of the result, set if the
;         result is negative.  the addressible bit ZOV is set
;         if Y is zero.
; destroys:    a
;=================================================================
SDIV8:
    mov     a, XA         ; read X and ...
    mov     b, YA         ; ... Y
    div     ab         ; divide X and Y
    mov     C, OV         ; get overflow flag
    mov     ZOV, C     ; save in ZOV.  (set if Y=0)
    mov     ZAQ, a     ; save result quotient
    mov     ZAR, b     ; save remainder

    jb      XS, SD8_2      ; get sign of X
    jb      YS, SD8_1
    clr     ZS
    ret
SD8_1:
    setb    ZS
    ret
SD8_2:
    jb      YS, SD8_3
    setb    ZS
    ret
SD8_3:
    clr     ZS
    ret

;=====================================================================
```
; **subroutine ADD16**

; 16-Bit Unsigned Addition

```
;
; input:    internal registers XH, XL, YH, and YL hold the high and
;           low bytes of the unsigned words
; output:   internal registers ZH and ZL hold high and low bytes of
;           the unsigned word Z=X+Y.  addressible bit ZOV is set if
;           the result (Z) is out of range (an external carry is
;           produced from the high byte)
; destroys: a, flags
;=====================================================================
ADD16:
    mov   a, XL        ; load X low byte into accumulator
    add   a, YL        ; add Y low byte
    mov   ZL, a        ; put result in Z low byte
    mov   a, XH        ; load X high byte into accumulator
    addc  a, YH        ; add Y high byte with the carry from low ...
                       ; ... byte operation
    mov   ZH, a        ; save result in Z high byte
    mov   ZOV, C       ; set ZOV if external carry
    ret                ; done


;=====================================================================
; subroutine SUB16
; 16-Bit Unsigned subtraction
;
; input:    internal registers XH, XL, YH, and YL hold the high and
;           low bytes of the unsigned words
; output:   internal registers ZH and ZL hold high and low bytes of
;           the signed word Z=X-Y.  addressible bit ZOV is set if
;           the result (Z) is out of range (if an external barrow is
;           produced)
; destroys: a, flags
;=====================================================================
SUB16:
    mov   a, XL        ; load X low byte into accumulator
    clr   C            ; clear carry flag
```

```
    subb     a, YL          ; subract Y low byte
    mov     ZL, a          ; put result in Z low byte
    mov     a, XH          ; load X high byte into accumulator
    subb     a, YH          ; subtract Y high byte with the barrow ...
                   ; ... from low byte operation
    mov     ZH, a          ; save result in Z high byte
    mov     ZOV, C     ; set ZOV if an external barrow is produced
    ret              ; done
;====================================================================
; subroutine SUB32
; 32-Bit Unsigned subtraction
;
; input:     internal registers X3, X2, X1, X0, and Y3, Y2, Y1, and Y0
;          hold the 32-bit unsigned integers
; output:     internal registers Z3, Z2, Z1, and Z0 hold the result
;          Z=X-Y.  addressible bit ZOV is set if the result (Z) is
;          out of range (if an external barrow is produced)
; destroys:     a, flags
;====================================================================
SUB32:
    mov     a, X0          ; load X low byte into accumulator
    clr     C              ; clear carry flag
    subb    a, Y0          ; subract Y low byte
    mov     Z0, a          ; put result in Z low byte
    mov     a, X1          ; repeat with other bytes...
    subb    a, Y1
    mov     Z1, a
    mov     a, X2
    subb    a, Y2
    mov     Z2, a
    mov     a, X3
    subb    a, Y3
    mov     Z3, a
    mov     ZOV, C         ; set ZOV if an external barrow is produced
    ret                 ; done
;====================================================================
```

```
; subroutine BMW
; Byte Multiplied by 16-Bit Word
;
; input:     internal register X holds the 8-bit unsigned integer.
;            internal registers YH, and YL hold the high and
;            low bytes of the unsigned word.
; output:    internal registers Z2, Z1 and Z0 hold the three result
;            bytes.  Z2 is the most significant byte, and Z0, the least
;            significant.  The result always fits in 3 bytes.
; destroys:  a, b, r0, flags
;=======================================================================
BMW:
    mov   psw, #0    ; select register bank 0
    mov   a, X       ; load X into accumulator
    mov   b, YL      ; load Y low byte into b register
    mul   ab         ; multiply
    mov   Z0, a      ; save result low byte
    push  b          ; push result high byte
                     ; this byte needs to be added to the low
                     ; byte of the product X*YH

    mov   a, X       ; load X into accumulator
    mov   b, YH      ; load Y high byte into b register
    mul   ab         ; multiply
    pop   0          ; pop r0 to recall X*YL high byte
    add   a, r0      ; add X*YL high byte and X*YH low byte
    mov   Z1, a      ; save result
    clr   a          ; clear accumulator
    addc  a, b       ; a = b + carry flag
    mov   Z2, a      ; save result
    ret              ; done
;=======================================================================
; subroutine MUL16
; Multiply two 16-Bit Unsigned Words
;
; input:     internal register XH, XL, YH, and YL hold the high and
```

```
;          low bytes of the two unsigned words X and Y.
; output:     internal registers Z3, Z2, Z1 and Z0 hold the four result
;          bytes.  Z3 is the most significant byte, and Z0, the least
;          significant.  The result always fits in 4 bytes.
; destroys:     a, b, r0, r1, flags
;=====================================================================
MUL16:
    lcall BMW          ; multiply XL with (YH:YL)
    mov     a, Z2
    push    acc        ; push ZL2
    mov     a, Z1
    push    acc        ; push ZL1
    mov     a, Z0
    push    acc        ; push ZL0

    mov     XL, XH     ; put XH in XL
    lcall   BMW        ; multiply XH with (YH:YL)

    mov     b, Z0      ; save ZH0 in b register
    pop     Z0         ; recall ZL0
    pop     acc        ; restore ZL1 in accumulator
    add     a, b       ; add ZH0 and ZL1
    mov     b, Z1      ; save Z1 in b register
    mov     Z1, a      ; save result
    pop     acc        ; restore ZL2 in accumulator
    addc    a, b       ; add ZH1 and ZL2 with carry from ZH0+ZL1
    mov     b, Z2      ; save ZH2 in b register
    mov     Z2, a      ; save ZH1+ZL2+C in Z2
    clr     a          ; clear accumulator
    addc    a, b       ; add ZH2 with carry from ZH1+ZL2
    mov     Z3, a      ; save result
    ret                ; done

    mov     a, X       ; load X into accumulator
    mov     b, YL      ; load Y low byte into b register
    mul     ab         ; multiply
```

```
    mov    Z0, a      ; save result low byte
    push   b          ; push result high byte
                      ; this byte needs to be added to the low
                      ; byte of the product X*YH

    mov    a, X       ; load X into accumulator
    mov    b, YH      ; load Y high byte into b register
    mul    ab         ; multiply
    pop    0          ; pop r0 to recall X*YL high byte
    add    a, r0      ; add X*YL high byte and X*YH low byte
    mov    Z1, a      ; save result
    clr    a          ; clear accumulator
    addc   a, b       ; a = b + carry flag
    mov    Z2, a      ; save result
    ret               ; done

;====================================================================
; subroutine DIV16
; 16-Bit Unsigned Division
;
; input:    internal registers XH, XL, YH, and YL hold the high and
;           low bytes of the unsigned dividend and divisor.
; output:   internal registers Z3 and Z2 hold high and low bytes of
;           the quotient of Z=X¸Y.  internal registers Z1 and Z0 hold
;           the high and low bytes of the remainder.  addressible bit
;           ZOV is set if Y=0, i.e., the result is out of range.
;
;           r1 and r0 store the high and low bytes of the partial
;           remainder.  r3 and r2 hold the partially computed quotient.
;
; calls:    SUB16
; destroys: a, r0, r1, r2, r3, r7, flags
;====================================================================
DIV16:
    mov    a, YH          ; get divisor high byte
    orl    a, YL          ; OR with low byte
```

```
    jnz     div_OK          ; divisor OK if not 0
    setb    ZOV             ; else, overflow
    ret
div_OK:
    mov     r1, XH          ; store dividend in r1, r0
    mov     r0, XL
    mov     XH, #0          ; clear partial remainder
    mov     XL, #0
    mov     r3, #0          ; clear partial quotient
    mov     r2, #0
    mov     r7, #16         ; set loop count

div_loop:
    clr     C               ; clear carry flag
    mov     a, r0           ; shift the highest bit of the dividend...
    rlc     a               ; ... into...
    mov     r0, a
    mov     a, r1
    rlc     a
    mov     r1, a

    mov     a, XL           ; ... the lowest bit of the partial ...
    rlc     a               ; ... remainder
    mov     XL, a
    mov     a, XH
    rlc     a
    mov     XH, a

    lcall   SUB16           ; attempt to subtract to see if the...
                            ; ... partial remainder is as large or...
                            ; ... larger than the divisor.

    mov     C, ZOV          ; get subtraction external barrow
    cpl     C               ; complement external barrow
    jnc     div_1           ; do not update partial remainder if no barrow
```

```
    mov     XH, ZH          ; update partial quotient
    mov     XL, ZL
div_1:
    mov     a, r2           ; add result bit to partial quotient
    rlc     a
    mov     r2, a
    mov     a, r3
    rlc     a
    mov     r3, a
    djnz    r7, div_loop

    mov     Z3, r3          ; put quotient in Z3, and Z2
    mov     Z2, r2
    mov     Z1, XH          ; get remainder, saved before the...
    mov     Z0, XL          ; last subtraction
    clr     ZOV             ; divisor is not 0
    ret                     ; done


;====================================================================
; subroutine DIV32
; 32-Bit Unsigned Division
;
; input:    internal registers X3, X2, X1, and X0 hold the 32-bit
;           unsigned dividend.  similarly, internal registers Y1 and
;           Y0 hold the high and low bytes of the unsigned divisor.
;
; output:   internal registers Z5, Z4, Z3, and Z2 hold the 32-bit
;           quotient of Z=X¸Y.  internal registers Z1 and Z0 hold
;           the high and low bytes of the remainder.  addressible bit
;           ZOV is set if Y=0, i.e., the result is out of range.
;
;           internal registers PR3, PR2, PR1, and PR0 store the
;           partial remainder.  r5, r4, r3, and r2 hold the partially
;           computed quotient.
;
```

```
; calls:     SUB16
; destroys:     a, r2, r3, r4, r5, r7, flags
;==============================================================
DIV32:
    mov     a, Y1           ; get divisor high byte
    orl     a, Y0           ; OR with low byte
    jnz     div32_OK        ; divisor OK if not 0
    setb    ZOV             ; else, overflow
    ret
div32_OK:
    mov     Y2, #0          ; high 16 bits of Y is 0
    mov     Y3, #0
    mov     PR3, X3         ; store dividend
    mov     PR2, X2
    mov     PR1, X1
    mov     PR0, X0
    mov     X3, #0          ; clear partial remainder
    mov     X2, #0
    mov     X1, #0
    mov     X0, #0
    mov     r6, #0
    mov     r5, #0
    mov     r4, #0          ; clear partial quotient
    mov     r3, #0
    mov     r7, #32         ; set loop count

div32_loop:

    clr     C               ; clear carry flag
    mov     a, PR0          ; shift the highest bit of the dividend...
    rlc     a               ; ... into...
    mov     PR0, a
    mov     a, PR1
    rlc     a
    mov     PR1, a
    mov     a, PR2
```

```
    rlc     a
    mov     PR2, a
    mov     a, PR3
    rlc     a
    mov     PR3, a

    mov     a, X0           ; ... the lowest bit of the partial ...
    rlc     a               ; ... remainder
    mov     X0, a
    mov     a, X1
    rlc     a
    mov     X1, a
    mov     a, X2
    rlc     a
    mov     X2, a
    mov     a, X3
    rlc     a
    mov     X3, a

    lcall   SUB32           ; attempt to subtract to see if the...
                            ; ... partial remainder is as large or...
                            ; ... larger than the divisor.

    mov     C, ZOV          ; get subtraction external barrow
    cpl     C               ; complement external barrow
    jnc     div32_1         ; do not update partial remainder if no barrow

    mov     X3, Z3          ; update partial quotient
    mov     X2, Z2
    mov     X1, Z1
    mov     X0, Z0
div32_1:
    mov     a, r3           ; add result bit to partial quotient
    rlc     a
    mov     r3, a
```

```
    mov     a, r4
    rlc     a
    mov     r4, a

    mov     a, r5
    rlc     a
    mov     r5, a

    mov     a, r6
    rlc     a
    mov     r6, a

    djnz    r7, div32_loop

    mov     Z5, r6          ; put quotient in Z3, and Z2
    mov     Z4, r5
    mov     Z3, r4
    mov     Z2, r3
    mov     Z1, X1          ; get remainder, saved before the...
    mov     Z0, X0          ; last subtraction
    clr     ZOV             ; divisor is not 0
    ret                     ; done
;====================================================================
; subroutine SADD16
; 16-Bit Signed (Two's Complement) addition
;
; input:     internal registers XH, XL, YH, and YL hold the high and
;            low bytes of the signed words in two's complement
;            convension
; output:    internal registers ZH and ZL hold high and low bytes of
;            the signed word Z=X+Y.  addressible bit ZOV is set if
;            the result (Z) is out of range (-32768 to 32767)
; destroys:  a, flags
;====================================================================
SADD16:
    mov     a, XL           ; load X low byte into accumulator
```

```
    add     a, YL          ; add Y low byte
    mov     ZL, a          ; put result in Z low byte
    mov     a, XH          ; load X high byte into accumulator
    addc    a, YH          ; add Y high byte with the carry from low ...
                           ; ... byte operation
    mov     ZH, a          ; save result in Z high byte
    mov     C, OV          ; get the overflow flag and ...
    mov     ZOV, C         ; ... transfer to bit ZOV
    ret                    ; done
```

```
;====================================================================
; subroutine SSUB16
; 16-Bit Signed (Two's Complement) subtraction
;
; input:     internal registers XH, XL, YH, and YL hold the high and
;            low bytes of the signed words in two's complement
;            convension
; output:    internal registers ZH and ZL hold high and low bytes of
;            the signed word Z=X-Y.  addressible bit ZOV is set if
;            the result (Z) is out of range (-32768 to 32767)
; destroys:  a, flags
;====================================================================
SSUB16:
    mov     a, XL          ; load X low byte into accumulator
    clr     C              ; clear carry flag
    subb    a, YL          ; subract Y low byte
    mov     ZL, a          ; put result in Z low byte
    mov     a, XH          ; load X high byte into accumulator
    subb    a, YH          ; subtract Y high byte with the barrow ...
                           ; ... from low byte operation
    mov     ZH, a          ; save result in Z high byte
    mov     C, OV          ; get the overflow flag and ...
    mov     ZOV, C         ; ... transfer to bit ZOV
    ret                    ; done
```

```
;==================================================================
; subroutine SMUL16
; multiplication of signed 16-Bit words in the Absolute Value /
; Sign Bit format
;
; input:    internal registers XH, XL and YH, YL hold the
;           magnitudes of the two signed words to be multiplied.
;           The addressible bits XS and YS are set if the
;           corresponding number is negative.
; output:   the absolute value of the result is placed in the 4
;           internal registers named Z3, Z2, Z1, and Z0.
;           Addressable bit ZS is set if the result is negative.
; calls:    MUL16
; destroys: a
;===================================================================
SMUL16:
    lcall   MUL16        ; multiply absolute values
    mov     C, XS        ; get sign of X

    jb      XS, SM16_2   ; get sign of X
    jb      YS, SM16_1
    clr     ZS
    ret
SM16_1:
    setb    ZS
    ret
SM16_2:
    jb      YS, SM16_3
    setb    ZS
    ret
SM16_3:
    clr     ZS
    ret

;==================================================================
; subroutine SDIV16
```

```
; division of signed 16-Bit words in the Absolute Value / Sign
; Bit format
;
; input:     internal registers XAH, XAL and YAH, YAL hold the
;          magnitudes of the two signed words.  the addressable
;          bits XS and YS are set if the corresponding number is
;          negative.
; output:     the absolute value of the result Z=X¸Y is placed in
;          the 4 internal registers named Z3, Z2, Z1, and Z0.
;          addressable bit ZS is set if the result is negative,
;          and ZOV is set if Y=0.
; calls:     DIV16
; destroys:     a
;==============================================================
SDIV16:
    lcall     DIV16        ; divide absolute values
    jb      XS, SD16_2      ; get sign of X
    jb      YS, SD16_1
    clr     ZS
    ret
SD16_1:
    setb    ZS
    ret
SD16_2:
    jb      YS, SD16_3
    setb    ZS
    ret
SD16_3:
    clr     ZS
    ret


;==================================================================
; subroutine FMUL
; multiply two floating point numbers
;
```

```
; input:     internal registers X2, X1, X0 and Y2, Y1, Y0 hold the
;            two floating point numbers to be multiplied.
; output:    the product Z = X ´ Y is saved in Z2, Z1, Z0.
; calls:     MUL16
; destroys:  a, b, r2, r3, r4
;=====================================================================
FMUL:
    mov    a, X1        ; get the sign bit plus 7 mantissa bits of X
    mov    C, acc.7     ; read sign bit
    mov    XS, C        ; save sign bit
    mov    a, Y1        ; get the sign bit plus 7 mantissa bits of Y
    mov    C, acc.7     ; read sign bit
    mov    YS, C        ; save sign bit

    jb     XS, FMUL_2   ; get sign of X
    jb     YS, FMUL_1
    clr    ZS
    sjmp   FMUL_4
FMUL_1:
    setb   ZS
    sjmp   FMUL_4
FMUL_2:
    jb     YS, FMUL_3
    setb   ZS
    sjmp   FMUL_4
FMUL_3:
    clr    ZS

FMUL_4:
    mov    r2, X2       ; save exponent of X in r2
    mov    r3, Y2       ; save exponent of Y in r3

    cjne   r2, #0, XNZ  ; see if X is zero
    mov    Z2, #0       ; if X=0, then Z=0
    mov    Z1, #0
    mov    Z0, #0
```

```
    ret                 ; done
XNZ:
    cjne   r3, #0, YNZ   ; see if Y is zero
    mov    Z2, #0        ; if Y=0, then Z=0
    mov    Z1, #0
    mov    Z0, #0
    ret                 ; done
YNZ:
    mov    a, r2         ; get exponent of X
    add    a, r3         ; add exponent of Y
    mov    b, #0         ; clear b register
    mov    b.0, C        ; move carry bit into b register
                        ; b register is high byte of sum of exponents
    clr    C
    subb   a, #128       ; subtract 128 from sum of exponents low byte
    mov    r2, a         ; save exponent in r2
    mov    a, b          ; get sum of exponents high byte
    subb   a, #0         ; subtract high bytes
    jnc    NoUNF         ; external barrow means underflow
    mov    Z2, #0        ; underflow...set Z=0
    mov    Z1, #0        ; it may be more appropriate in some
    mov    Z0, #0        ; applications to set ZOV and signal en error
    ret                 ; done
NoUNF:
    jz     NoOVF         ; if high byte is 1 then overflow
    setb   ZOV           ; set overflow error flag
    ret                 ; done

; ---    if the program reaches NoOVF, the exponent is in r2 ---
; --- and the sign is in ZS.  next multiply mantissas     ---
NoOVF:
    mov    a, X1         ; get X1
    setb   acc.7         ; set the implied MSB=1
    mov    X1, a         ; save completed mantissa of X
    mov    a, Y1         ; get Y1
    setb   acc.7         ; set the implied MSB=1
```

```
    mov     Y1, a           ; save completed mantissa of Y

    lcall MUL16             ; multiply the mantissas

    mov    a, Z3            ; get mantissa high byte
    jb     acc.7, XYOK      ; mantissa is OK if MSB is 1

; --- if MSB of XY is 0, then the mantssa is shifted left ---
; --- and the exponent is incremented -----------------------
    inc    r2               ; this may result in an overflow
    cjne   r2, #0, EXP_OK
    setb   ZOV              ; overflow
    ret

EXP_OK:
    clr    C
    mov    a, Z1
    rlc    a
    mov    a, Z2
    rlc    a
    mov    Z0, a
    mov    a, Z3
    rlc    a

    mov    C, ZS
    mov    acc.7, C         ; sign bit in place of the implied '1'
    mov    Z1, a
    mov    Z2, r2
    ret
XYOK:
    mov    Z0, Z2           ; mantissa low byte
    mov    a, Z3            ; mantissa high byte
    mov    C, ZS
    mov    acc.7, C         ; sign bit in place of the implied '1'
    mov    Z1, a
    mov    Z2, r2           ; get exponent
```

```
ret ; done
```